A2Foodies: The Consistency of Restaurants Metrics
Lennox Thomas and Nicole Surcel

Link to GitHub Repository: https://github.com/lennoxbt/SI206FinalProject.git

Initial Goals

Initially, we planned to collect data concerning the top 100-rated restaurants in the Ann Arbor area, specifically concerning the restaurants' names, locations, ratings, price ranges, and types of food they serve. We intended to collect a list of restaurants from *Yelp* and then obtain data from the *Yelp* API of the restaurants' ratings. From this, we would then create a total of two visualizations: the first comparing the highest-rated restaurants to the most common restaurant locations and the second depicting the relationship between the price ranges of restaurants and their ratings on *Yelp*.

Problems

Despite our initial goals, we altered aspects of our project due to conflicts we experienced while working on the respective code.

Problem #1: To begin with, we found Ann Arbor to be too confining of a location to collect data from; in particular, it did not offer rankings of 100 or more popular restaurants within the area. Additionally, we attempted to use a list of top restaurants throughout Michigan, yet there once again wasn't a consistent list we could find.

Solution: To resolve the limited data we were receiving, we decided to expand to assessing restaurants across the United States. This allowed us to gather plentiful information concerning 100 dining locations, as well as their locations, ratings, price ranges, and types of food they offered.

Problem #2: When gathering restaurant ratings from the *Yelp* API, certain places did not receive sufficient reviews in comparison to others; there was also not a coherent list of the top 100 restaurants in the United States.

Solution: This resulted in us turning to a new website, *OpenTable*, where we were able to find one consistent list of highly-rated restaurants in the United States. We then took the data from *Open Table* and compared it to the information offered by the *Yelp* API to add diversity to the information we collected and expand on our final visualizations.

Problem #3: When gathering data from the *Yelp* API and attempting to place it within our "Restaurants" table, we struggled with generating values for the "Yelp_Rating" column; in particular, whenever the restaurantData.db was opened in SQLite, the column for *Yelp* ratings would produce results of "NULL" for all 100 restaurants.

Solution: To resolve this issue, we generated the following execution: cursor.execute('UPDATE Restaurants SET Type_ID=(select Type_ID from Types WHERE Types.Type=Restaurants.Type)'). This allowed us to access the types of restaurants within the "Restaurants" table and assign their "Type_ID" to the value designated to the "Type" column within our "Types" table.

Key Problems and Solutions After 04/26 Grading Session

Problem #4: One primary issue we faced throughout the completion of this project was ensuring that 25 or less data items were collected/stored within our "restaurantData.db" at a time.

Solution #4: We first made a text file ("count.txt") to hold our count values as the yelp.py processed with each run. In particular, we made use of the *Yelp* API (accessing the "Yelp_Rating" and "Yelp_Price" columns within our database) to execute our code to 25 items into the "restaurantData.db" each time we ran the yelp.py file.

Problem #5: Another primary issue we faced throughout the completion of this project was ensuring that there wasn't duplicate data collected/stored within our "restaurantData.db". In particular, the "Type" of each restaurant was included as a column in both our "Restaurants" table as well as our "Types" table.

Solution #5: We removed the "Type" column from our "Restaurants" table and instead used the yelp.csv to create a dictionary with key-value pairs from the "Type_ID" column within our "Restaurants" and "Types" tables. This then allowed us to alter our SELECT * FROM and JOIN statements within yelp.py and visualizations.py to connect the restaurant type in "Restaurants" with its respective ID from our "Types" table.

Solutions and Accomplishments

Ultimately, we collected a list of the top 100 restaurants in the United States (25 items at a time) from an *OpenTable* website and then obtained data from the *Yelp* API of the restaurants' ratings and prices to compare to their ratings and prices on *OpenTable*. From this, we then created a total of four visualizations: the first is a bar graph comparing the restaurant types to the average of their ratings (per type) on *OpenTable*; the second is a bar graph comparing the restaurant types to the average of their prices (per type) on *OpenTable*; the third is a pie chart depicting the percentages of restaurant types in the list of 100 restaurants present; and the fourth is a pie chart depicting the percentages of restaurant prices on *OpenTable* in the list of 100 restaurants present.

Calculations

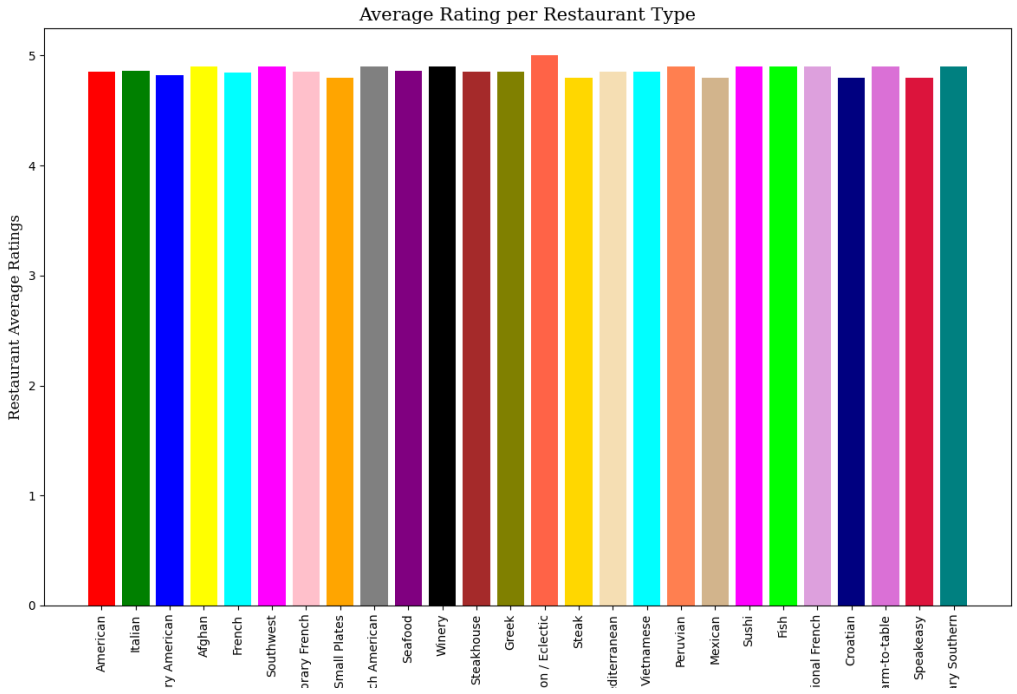
The following figure shows the output of the yelp.py file, which contains the calculated averages of the *OpenTable* and *Yelp* ratings combined for each restaurant that was not missing any metric data. The following lines shown below can be seen in the file attached to the GitHub

repository, titled “yelp.csv”. The first value is the restaurant name, while the second value represents the average of the *OpenTable* and *Yelp* ratings for each restaurant within the list.

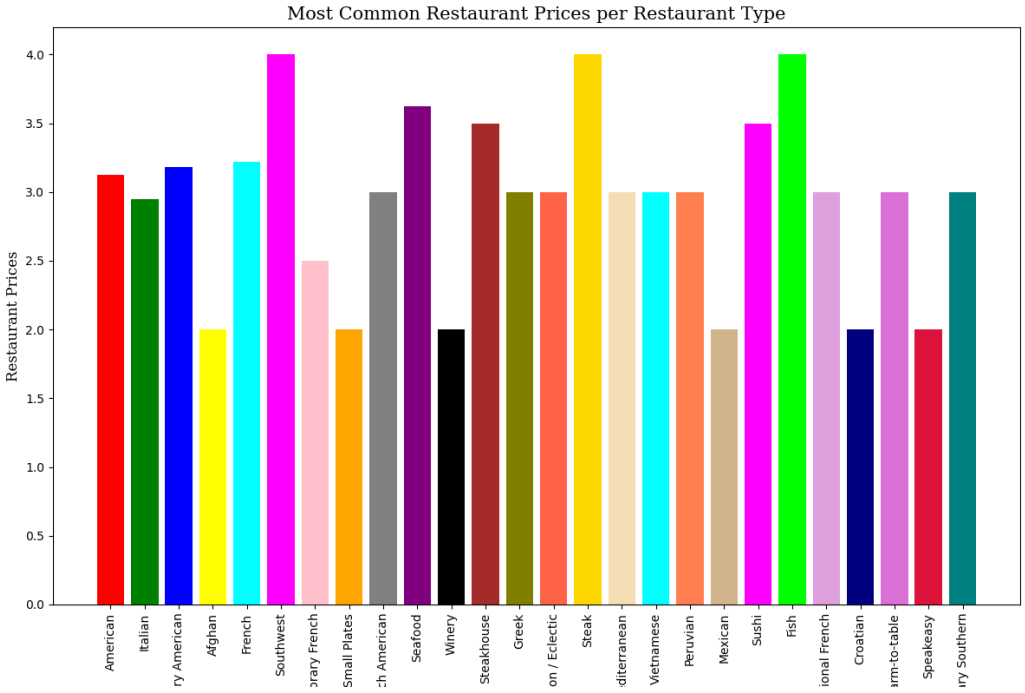
yelp.csv

```
Restaurant Name,Restaurant Average Rating
360 Bistro,4.2
Anjelica's Restaurant,4.5
Arethusa al Tavolo,4.7
Artisan,4.7
Bistro Aracosia,4.7
Bites & Bubbles,4.25
Blue Ridge Grill,4.45
Bonnell's Fine Texas Cuisine,4.45
Bouchard Restaurant and Inn,4.7
Buccan,4.7
Café Ba-Ba-Reeba,4.4
Cafe Monarch,4.7
Cafe Provence,4.7
CDM Restaurant,4.65
Cesarina,4.45
Dominick's Steakhouse,4.7
Eddie V's - Orlando,4.65
Elia Authentic Greek Taverna,4.65
Fat Canary,4.65
Filomena Ristorante,4.4
Flight Restaurant & Wine Bar,4.7
Geronimo,4.75
GW Fins,4.7
Hannas Prime Steak,4.65
House of Prime Rib,4.45
"Joe's Seafood, Prime Steak & Stone Crab - Las Vegas",4.65
Kokkari Estiatorio,4.7
La Bistecca Italian Grille,4.45
La Nouvelle Maison,4.65
Lahaina Grill,4.7
Le Colonial - Chicago,4.4
Le Diplomate,4.65
Le Yaca - Virginia Beach,4.7
Llama Restaurant,4.7
M Sushi,4.7
Majorelle,4.65
Mama's Fish House,4.7
Mamma Maria,4.4
Mot Hai Ba - Lakewood,4.45
O-Ku - Nashville,4.7
Ocean 44,4.7
Ocean Prime - Tampa,4.65
Our Mom Eugenia - Great Falls,4.65
Parc,4.4
Paris 7th,4.45
Pepp & Dolores,4.7
Portosole,4.6
Red Ash Italia,4.45
Riccardo Enoteca,4.45
RL Restaurant,4.45
Rose Mary,4.65
Rosina's Italian Restaurant,4.6
Rosmarino Osteria Italiana,5.0
Ruth's Chris Steak House - Baton Rouge,4.75
Saloon Restaurant,4.7
Salt,3.9
Salum Restaurant,4.4
Satchel's on 6th,4.4
Savour,4.95
Sergio's Ristorante,4.7
Sotto,4.45
South of Nick's - Laguna Beach,4.65
State of Grace,4.7
Steak 44,4.7
Steak 48 - Houston,4.7
Terzo,4.4
The Boathouse,4.6
The Crossing,4.7
The Dock,4.75
The Grand Tavern,4.7
The House of William & Merry,4.7
The Metro Wine Bar & Bistro,4.7
The Original Spence Cafe,4.65
The Stables,4.65
Truluck's - Ocean's Finest Seafood & Crab - Naples,4.65
TwentyTwenty Grille,4.5
Uchiko Austin,4.7
Vargas Cut & Catch,4.7
Washington Inn,4.65
West Rose,4.65
Wildflower - Denver,4.95
```

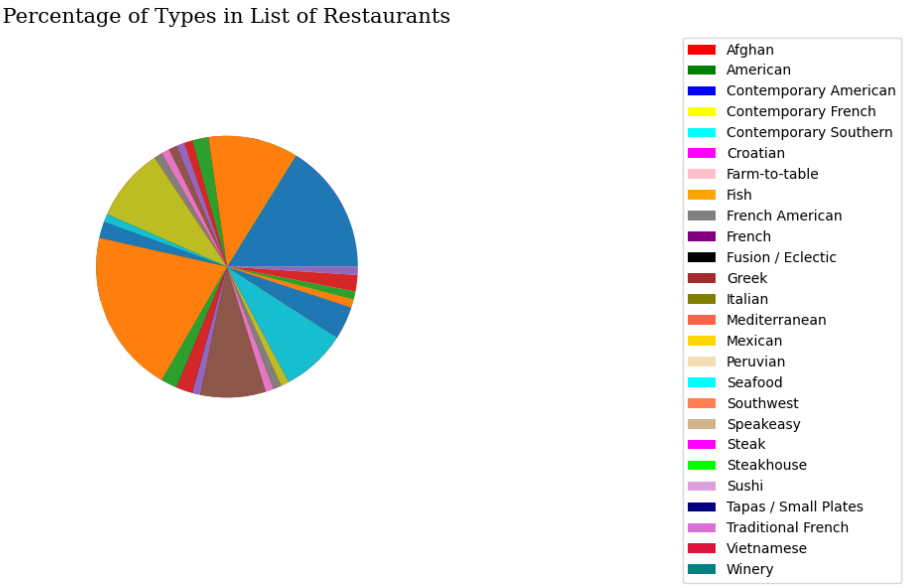
Visualizations



Description: This function takes in a dictionary which contains restaurant types as the keys and average ratings as the values. It also takes in the preferred filename of the graph which will be returned. The function uses the dictionary to create and decorate a bar chart, which will compare the average ratings across restaurant types. The function will output a jpeg file, with the preferred name that was passed into the function.



Description: This function takes in a dictionary which contains restaurant types as the keys and locations as the values. It also takes in the preferred filename of the graph which will be returned. The function uses the dictionary to create and decorate a bar chart, which will compare the prices across restaurant types. The function will output a jpeg file, with the preferred name that was passed into the function.



Description: This function takes no parameters. The function connects to our database and uses a SELECT statement to select all the columns from the Restaurants table. It then iterates through every row in the Restaurants table to track how many restaurants belong to each category type. It then creates a pie chart depicting the breakdown of percentage for each category (the category is the Type of Restaurant).

Description: This function takes no parameters. The function connects to our database and uses a SELECT statement to select all the columns from the Restaurants table. It then iterates through every row in Restaurants to track how expensive each restaurant is based on type, ranging from one dollar sign ($) to four ($$$$). After this, we created a pie chart depicting the breakdown of percentages for each category, $ being least expensive and $$$$ most expensive.

Code Instructions

Ensure all CSV and database files are deleted before running the code; the only files that should be in the folder are SI 206 Final Write-Up.pdf, README.md, opentable.py, yelp.py, visualizations.py, count.txt, Figure_1.png, Figure_2.png, Figure_3.png, and Figure_4.png. Run each file once in the following order:

1) opentable.py
2) yelp.py
3) visualizations.py

Documentation

opentable.py

```
def restaurantlst(url):
  # This function takes in a url from the "OpenTable" website, and creates a BeautifulSoup Object to parse through
  # the site's HTML. The function returns a list of tuples which include the restaurant name as well as the restaurant's location.

def write_csv(data, filename):
  # This file takes in a list of tuples and csv filename as input. It then iterates through the list of tuples
  # to write multiple rows within the csv, outputting a csv file containing each restaurant name and restaurant location.

def setUpDatabase(db_name):
  # This function simply takes in a string as input which contains the preferred database file name, and
  # returns the cursor and connection to the created database.

def setUpRestaurantsTable(cur, conn, restaurantlst, x = 0):
  # This takes a database cursor and conneciton, list of restaurant names, restaurant type, and optional
  # argument, which specifies the starting position of the database ID. The function creates a table, Restaurants,
  # within the passed database and inserts each restaurant in restaurantlst, along with its corresponding ID, location, and type.

def setUpTypesTable(cur, conn, typelst):
    # This function takes in a databse cursor and connection, as well as a list of restaurant type. It then creates
    # a table, Types, within the database, along with its corresponding ID number.

def main():
    # This function calls all of the above functions: writing the "restaurant.csv" file, setting up the database,
    # defining the restaurant types, grabbing the lists of restaurants, and setting up the database tables ("Restaurants" and "Types").
```

yelp.py

```
<yelp.py functions>
def getYelpRatings():
    # This function gathers data from the Yelp API, 20 items at a time. It accesses the restaurant names in the restaurant.csv file in order to
    # pull multiple restaurant metrics from the api. This pulled data is then placed into the tables of the restaurantData.db
    # database.

def yelp_csv(filename):
    # This function takes in a csv filename as input, and selects data from the joined tables, Restaurants and Types.
    # This data is then used to calculate the average of the OpenTable and Yelp ratings for each restaurant.
    # This calculation is then output into the yelp.csv file, along with the restaurant name, and average rating between Yelp and OpenTable.

def main():
    # This function calls the above functions, getYelpRatings and yelp_csv. In order to gather enough data from
    # the Yelp API, getYelpRatings is called within a for loop, multiple times.
```

visualizations.py

```
<visualizations.py functions>
def getTypeRatingData(db_filename):
    # This function takes in a database filename and restaurant type as its input. It then creates a connection and cursor to the database,
    # and selects several values to calculate the average rating for each restaurant type in the database. It then outputs this
    # information in a dictionary with the types as the key, and the ratings as the values.

# Visualization #1
def barchart_restaurant_ratings(restaurant_dict1, name):
    # This function takes in a dictionary which contains restaurant types as the keys and average ratings as the values. It also takes
    # in the preferred filename of the graph which will be returned. The function uses the dictionary to create and decorate a bar chart,
    # which will compare the average ratings across restaurant types. The function will output a jpeg file, with the preferred name
    # that was passed into the function.

def getTypePriceData(db_filename):
    # This function takes in a database filename and restaurant type as its input. It then creates a connection and cursor to the database,
    # and selects several values to calculate the average price for each restaurant type in the database. It then outputs this
    # information in a dictionary with the types as the key, and the ratings as the values.

# Visualization #2
def barchart_restaurant_prices(restaurant_dict2, name):
    # This function takes in a dictionary which contains restaurant types as the keys and locations as the values. It also takes
    # in the preferred filename of the graph which will be returned. The function uses the dictionary to create and decorate a bar chart,
    # which will compare the prices across restaurant types. The function will output a jpeg file, with the preferred name
    # that was passed into the function.

# Visualization #3
def piechart_restaurant_types():
    # This function takes no parameters. The function connects to our database and uses a SELECT statement to select all the columns
    # from the Restaurants table. It then iterates through every row in the Restaurants table to track how many restaurants belong to each
    # category type. It then creates a pie chart depicting the breakdown of percentage for each category (the category is the Type of Restaurant).

#Visualization #4
def piechart_restaurant_prices():
    # This function takes no parameters. The function connects to our database and uses a SELECT statement to select all the columns
    # from the Restaurants table. It then iterates through every row in Restaurants to track how expensive each restaurant is based on
    # type, ranging from one dollar sign ($) to four ($$$$). After this, we created a pie chart depicting the breakdown of percentages
    # for each category, $ being least expensive and $$$$ most expensive.
```

Resources

Date	Issue Description	Location Resource	Result
03/27/2022	We hoped to review the most quick and efficient method to generate an effective bar graph, specifically with data from our "Restaurants" and "Types" tables.	https://www.analyticsvidhya.com/blog/2021/08/understanding-bar-plots-in-python-beginners-guide-to-data-visualization/	The website demonstrated the primary methods of developing a bar graph, specifically with additional features we hoped to add, such as a large number of tickers, bar colors, and establishing a large table size.
04/03/2022	We hoped to expand the color variety of the bars and sectors within the bar and pie graphs to 26 different colors; this would	https://matplotlib.org/3.5.0/gallery/color/named_colors.html	This website provided a list of a multitude of colors to select from when assigning them to a bar and pie graph. As

	represent the 26 different restaurant types of the restaurants on the *OpenTable* list.		a result, a greater variety of sections were highlighted within our visualizations.
04/10/2022	We intended to pull the 'rating' class for each restaurant within the *OpenTable* website.	https://pytutorial.com/get-aria-label-beautifulsoup	This tutorial demonstrated the use of the ['aria-label'], which we needed to declare in the web-scraping of the *OpenTable* website in order to access the *OpenTable* rating for each restaurant listed.
4/16/2022	We intended to convert the *OpenTable* 5-star ratings from a string ("5 s") to a float (5.0).	https://www.geeksforgeeks.org/convert-string-to-float-in-python/	This website provided a way to ensure all *OpenTable* ratings were presented as strings on our "Restaurants" table (*ex: 4.8, 4.9, 5.0*).
04/20/2022	We struggled with generating values for the "Yelp_Rating" column when gathering data from the *Yelp* API and attempting to place it within our "Restaurants" table.	https://chartio.com/resources/tutorials/how-to-alter-a-column-from-null-to-not-null-in-sql-server/	This tutorial provided us with a method to access the types of restaurants within the "Restaurants" table and assign their "Type_ID" to the value designated to the "Type" column within our "Types" table.